Process Book

Rehan Mulakhel, Noemi Romano, Raja Soufi
Department of Computer Science, EPFL Lausanne, Switzerland

I. Introduction

Over human history, thousands and thousands of meteorites fell to the Earth ground; chunk of rock and metal disagreggating in the atmosphere, hitting the ground and causing sometimes vast disasters from natural and economical point of view.

Where are the points which have the most chances to be hit? Can we trust the data gathered? Why do some places have less chances to be receive rocks coming from space?

The goal of this project is to visualize this fascinating phenomenon occurring over the last centuries by means of georeferenced location of the impacts recorded by the *Meteoritical society*[1]. In this visualization, our principal aim is to give a general overview of the spatio-temporal evolution of this natural phenomenon, emphasizing on the user experience and the exploration of the data. This visualization targets a general public who has not strong knowledge in the field.

II. Data

The data come from the NASAs Open Data Portal and have been downloaded from Kaggles platform. The data were collected by The Meteoritical Society and contain information on all of the known meteorite landings. The dataset includes the following fields:

`name`
: The name of the meteorite (typically a location, often modified with a number, year, composition, etc).

`id`
: The unique identifier for the meteorite.

`nametype`
: One of: – `valid`: a typical meteorite – `relict`: a meteorite that has been highly degraded by weather on Earth.

`recclass`
: The class of the meteorite; one of a large number of classes based on physical, chemical, and other characteristics.

`mass`
: The mass of the meteorite, in grams.

`fall`
: Whether the meteorite was seen falling, or was discovered after its impact; one of: – Fell: the meteorite's fall was observed – Found: the meteorite's fall was not observed.

`year`
: The year the meteorite fell, or the year it was found (depending on the value of fell).

`reclat`
: The latitude of the meteorite's landing.

`reclong`
: The longitude of the meteorite's landing.

`GeoLocation`
: The parentheses-enclose, comma-separated tuple that combines `reclat` and `reclong`.

Rows containing `NaN` values or presenting a years value smaller than 1800 and bigger than 2013 will not be considered in our visualization project. In addition, some of the entries having coordinates values equal to 0 —referring to meteorites found in Antarctica of which coordinates were not given— will not be considered. The data have been cleaned with `R` language.

After the cleaning, we end up with a final number of $31,705$ over $45,716$ entries.

III. Tools

The visualization is displayed in a browser for usability reasons: users do not need to download anything. This forces the application to be split into a front-end and a back-end.

A. Front-end

The main visualization of the globe and falling meteorites is done with Three.js. The rest of the visualizations (timeline, class statistics, etc.) were done with D3 and viz.js.

jQuery and Bootstrap were also used.

B. Back-end

Most files necessary for displaying the web page are served using GitHub pages (the libraries being fetched from various CDNs except for viz.js).

All the data related to meteorites and countries is saved on Google Drive and served using a simple Google Apps Script web app, which also provides a few endpoints to get filtered or grouped data (see here).

[1]Meteoritical society: http://www.meteoriticalsociety.org/

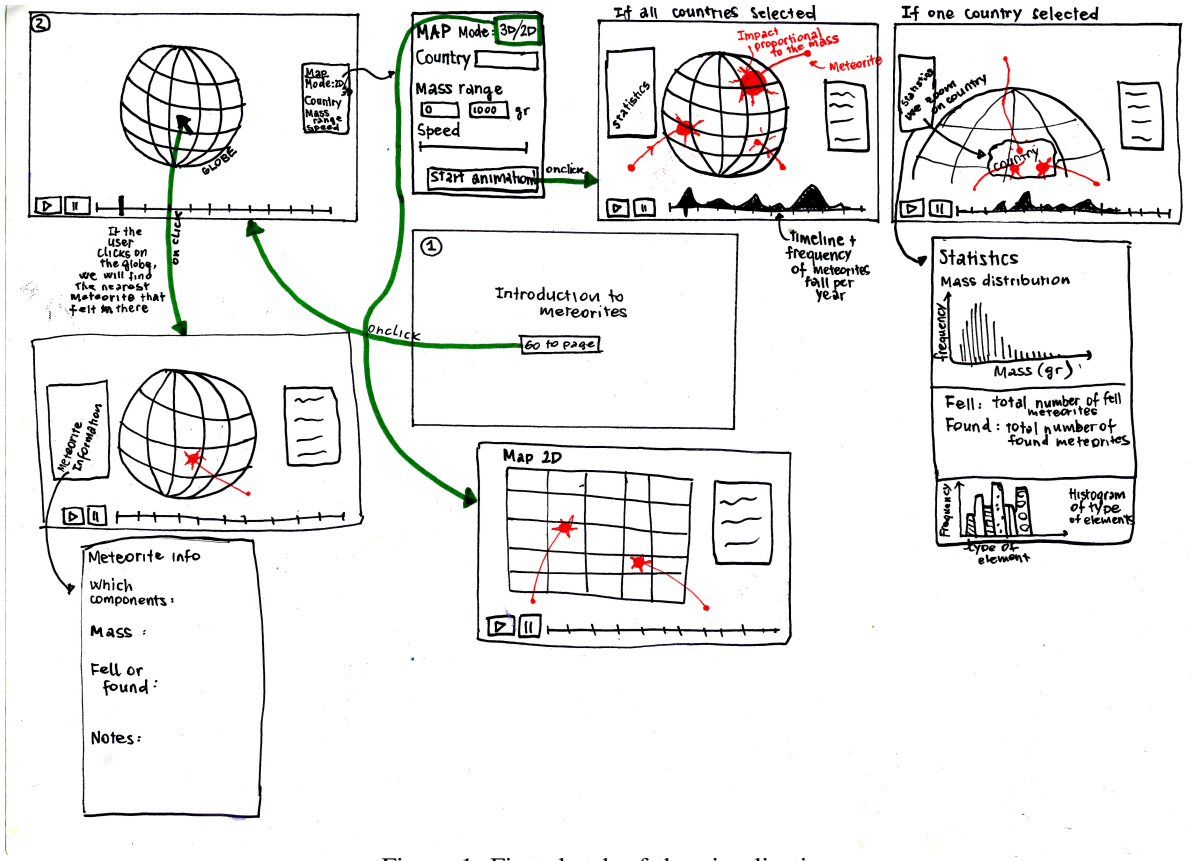
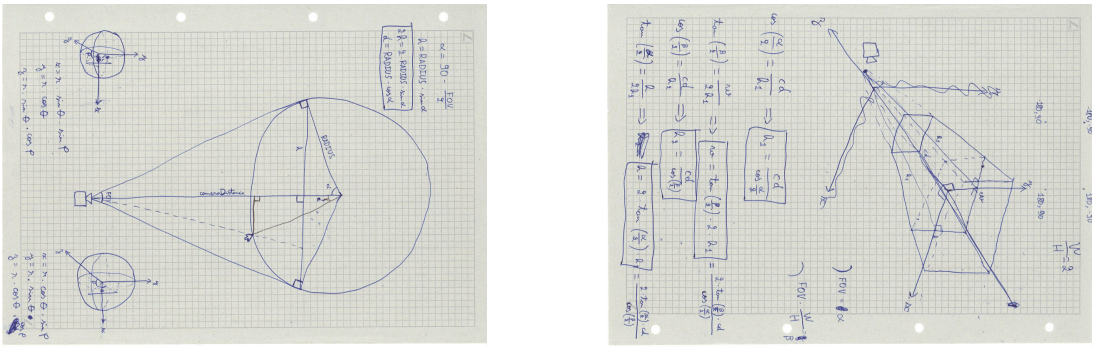


Figure 1: First sketch of the visualization



(a) Sketch of camera FOV for globe

(b) Sketch of camera FOV for 2D map

Figure 2: Sketch of camera Field of View (FOV)

IV. Evolution of visualization

The first step was to figure out which visualization could be appropriate to visualize the spatial distribution of the meteorites' impacts. A globe or a map with the georeferenced impacts as static points came to our mind, but the UX would have been limited. That's why we decided to animate the trajectory of the meteorite falls, in order to give a realistic dimension to the data. In order to enforce the realistic dimension we prioritized the 3D visualization, hence a globe has been chosen to visualize the geographical space. To give a more global point of view of the spatial distribution of this phenomena, the transition from the globe to a 2D map has been also taken into account.

Having the coordinates of the impacts, it was crucial to get the country where the meteorites fell. Getting access to the country implies then a new category of classification, leading to a local and easier exploration of the data. The local dimension gave us new ideas for the data representation:

- Visualization of the country related statistics

- A search-bar where the user can enter the country wished
- The interaction user-globe that allows to click over a country and see the meteorites falling for the country selected
- Histogram of the type of meteorite by country

A time-line was by far the easiest way to visualize the temporal dimension. In order to add an interaction to it, we decided to allow the selection of an interval of time to the user. The animation of the meteorites falls will then be performed in the interval of time selected and activated/deactivated with a start/pause button.

The first sketch of the visualization, which was drawn before starting the code, can be seen in figure 1.

A. Globe

Exploring the Mike Bostock's Blocks [5] we discovered many tutorial allowing the creation of a globe with `d3.js` [6, 7]. These works are performed with the `d3.geo` library that allows an easy way to work with geographic information and different projections.

We then created the globe, but a sticking point came up: how to handle the meteorites' animation with `d3`?

We then explored other javascript libraries that could handle in a simpler way the animation of 3D objects and we figured out that the `three.js` library [3] is the most suitable one. In fact, this library is a framework build on top of WebGL which makes it easier to create 3D graphics in the browser. Another reason why we chose this library is because we were not obligated to move the objects themselves, but just moving the main camera around the globe.

We inspired our globe from another Mike Bostock example [2] that shows how to import `geoJSON` files in a `three.js` scene. The `three` library doesn't read automatically objects with geometry as `d3` does. First of all, the `geoJSON` file containing the countries borders' multilinestring [1] coordinates are re-projected into spherical projection. Then, all the starting and ending vertices of the multilinestring have to be pushed in a `three` geometry and a line has to be drawn between all these couples of vertices.

A white sphere was added as well in order not to see through the globe. In order to have a good view of the globe and the 2D map, we needed to adapt the main camera FOV and distances (Figure 2).

After that the camera was well adjusted, we ended up with the globe illustrated in figure 3.

Figure 3: Globe with `three.js`

In order to add an interaction to the globe, the user can zoom-in, zoom-out, and drag the globe. The 2D map has also been taken into account, but being our first aim to make this visualization the more realistic as possible, we finally decided to keep only the globe.

B. Meteorites' country

The original data set provided by Kaggle did not contain any information on countries name. The only way to map a country to a landing was through the geo coordinates. Since the `csv` file contains many thousands rows, it was impossible to do it manually. This lead us to write a script in python in order to fetch countries based on the tuple (latitude, longitude).

Google Maps, which is likely to be the most famous web app for location, has its services limited to users to prevent DDoS[2] attack. According to their documentation, the number is limited to a "a few thousands" requests per day per person. This could have taken more than one week to get our new data! Clearly, we had to find a solution. That's why we dropped the idea of Google Maps and turned towards Open Street Map, an open source alternative.

Open Street Map also has a system to prevent DDoS attack. But it does not have a limited number of requests per day. Actually their api sends the error 429 which stands for "too many requests", requiring the script to block for several minutes. Eventually the script was run on a server for more than thirty hours.

We created automatically countries' code, but they did not mean anything if we ever wanted to analyze with other

[2]Distributed Denial of Service

countries' data available. Hence, we redo this process, but this time with Quantum GIS, an open source GIS[3] software. In this case, we just joined by location the countries vector layer containing the codes published by the International Organization for Standardization (ISO) to our meteorites falls points. This software gave us a general idea of the spatial distribution of the meteorites (Figure 4).

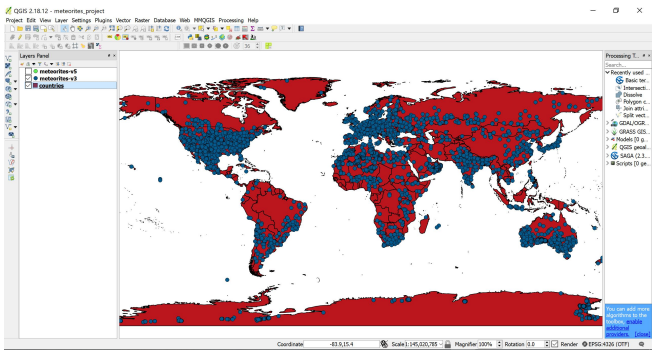


Figure 4: Meteorites points in Quantum GIS

C. Time-line

The first idea was to create an animation close to a video. The starting and the ending moments were supposed to be parameters. The inspiration came from [4], parts of the code were taken and adapted to the latest version of ES6[4]. There was a plot of the number of meteorites group by year inside the time-line.

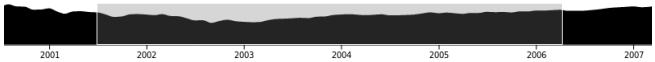


Figure 5: Time line with a brushing system. Here with fake data.

Selecting the time range was not as user friendly as expected. Instead, a flag with the 'current' year (of the animation) is moving like a cursor which can be dragged and dropped. In this way, only one button to start and stop is needed.

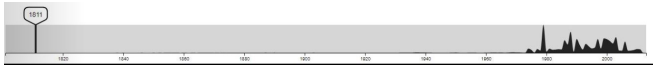


Figure 6: Time line with a flag for the year being in the animation.

The data contains date from 860 to 2016. The plot in figure 6 shows that most meteorites hit the Earth recently. This is unlikely to be the reality. We can interpret it like humans have almost no information on the subject until the 19th century. This information leads us to filter from 1801 to 2016.

We had the temporal evolution of the meteorites falls but still did not give any information about the masses of the meteorites. We then decided to add to the initial timeline the average mass per year (Figure 7a,7b).

The last step was to add labels for vertical axes because the two plots are not intuitive on their own unlike the time axes.

D. Meteorites animation

After settling up the globe scene, the meteorites animation needed to be done.

Our first idea was to implement a user interaction with the meteorites, where the user could choose the origin of the meteorites and launch it. This idea would have enforced the UX but it was difficult to choose which meteorite has to be launch and it was far from our principal idea to represent the spatio-temporal evolution of this phenomena.

Another idea was to find in the literature the real origin point of the meteorites, but this idea was soon abandoned because the time scale of the meteorites was per year and we did not know what was the rotation axis when the meteorites fell and it was challenging to place the other asteroids or planets in our 3D scene.

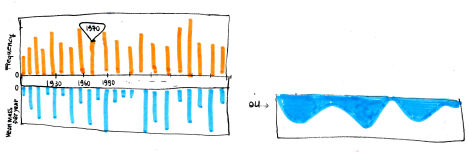
From the beginning, the main idea of the animation function is to take as parameters the destination geographical coordinates and the mass of the meteorite, give it a random origin point and update its position with a loop until the destination point. The mass was considered as well so that the size of the impacts of the meteorites are proportional to it.

In order to make the meteorites impacts as realistic as possible, we just created an additional geometry in the position of the impact and simply expanded it and changed its opacity with a loop. The meteorite impact is illustrated in figure 8.

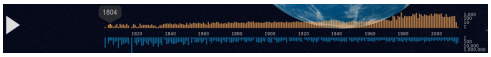
Figure 8: Meteorite impact

The first animation called a function that scheduled other

[3]Geographic Information Systems
[4]ECMAScript 6



(a) Sketch of time line with the frequency of the meteorite falls per year on the top and average mass per year on the bottom



(b) Time line with plots of frequencies (top top) and average mass (bottom) for each year.

Figure 7: The evolution of the time line from the paper to the final version.

hundreds functions which at each meteorite launching incremented the position of the meteorite. This method lead sometimes to some bugs in the animation due to the high CPU cost.

The second and final attempt is now calling a function that add the meteorites to an array and a separated loop iterates this array to increment the position of each meteorites. This method gave an important improvement in the animation and allowed to visualize thousand and thousand of meteorites without any bug.

Furthermore, we decided to add a light object to the meteorite to improve the realistic falls (FIgure light)

Unfortunately, this option was too CPU expensive and after two meteorites falls the visualization crashed.

E. Search bar

Users may want to choose and change some parameters like the speed or the dimension of the map (2D vs 3D). The initial version is shown in figure 9. The opacity was set to 0.5 in order to let the map (which was not ready at that time) be visible. When the mouse was hover the box, the transparency property was removed as long as the mouse was not outside of it.

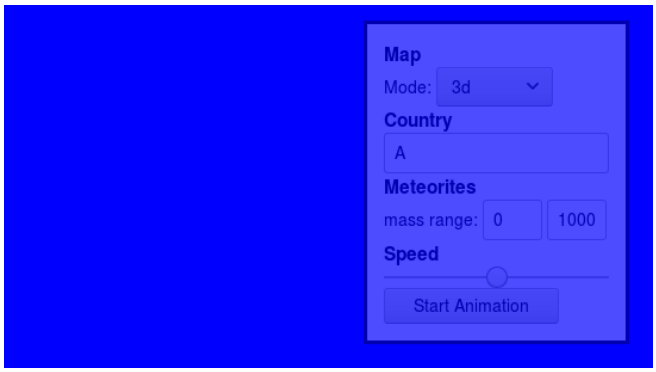


Figure 9: The transparent box of parameters of the animation.

The animation of the meteorites falling from the space can be done without much effort in a 3d world. Unfortunately, the 2d version does not allow it. That's why this option was removed.

Since the visualization is intended to a general public, the range of the mass is useless. This option was also removed.

What about the animation speed? Our first feeling was to create something constant over the time. The sparse number of landing per year makes it impossible. Let's assume that we set a constant λ for the speed. Since the frequencies of the number of meteorites increases, at some point the number of meteorites in the visualization will reach a threshold which will make the script crashing. If we decrease the value of λ, then animation would be to slow when the frequency is low. Therefore, the only way to do it is to make λ changing depending on the frequency. Thus, the cursor for the `speed` value was removed too.

Eventually we only kept the country input.

F. Meteorites classification

At the beginning, the meteorite classification was not taken into account because we did not have any knowledge in the domain, and targeting a general public, it would have been an irrelevant information to present. Going deeper in the domain, we discovered that the type of meteorites depends on the chemical elements of which they are composed by. The attribute `recclass` presents more than 300 unique classes of meteorites that belong to a defined hierarchy well presented in Weisberg et al. [11].

Going up in the nodes of the hierarchy, we ended up with three main classes that describe the material of the meteorites: *stony*, *stony-iron* and *iron*.

We firstly checked on the name of the unique classes and classify them by looking at the first letter of `reclass`. The classification was that easy because the *stony-iron* meteorites have only two sub-classes (*Pallasite* and *Mesosiderite*), and the *iron* meteorites are always classified as *Iron+code*. The *stony* class contains much more sub-classes. Giving that *iron* and the *stony-iron* meteorites contain a limited number of sub-classes, we just checked if the remaining meteorites classes were actually *stony* (and it was the case!) and assigned to them this class. This simple classification could then draws more attention of the user and make him more familiar with this domain. Indeed, we did not want to go much deeper in the details of the classes because it could have been useless for the public targeted.

Concerning the visualization, the first idea that came to

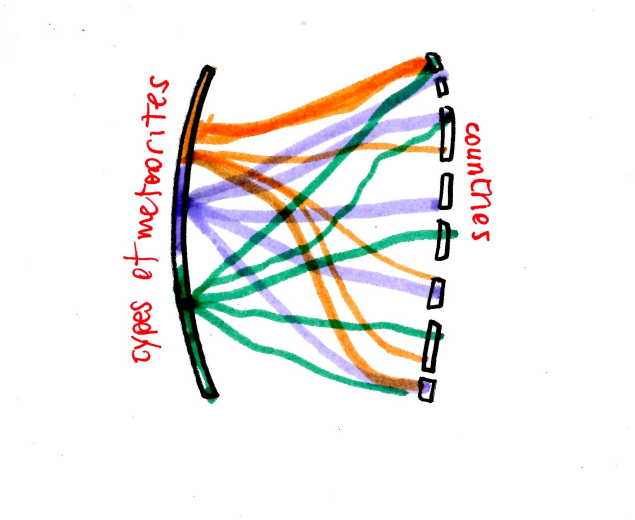


Figure 10: First sketch of the flows between types of meteorites and countries

our mind was to make a simple bar-chart for every country selected (Figure 1).

We soon abandoned this idea because we could have had only a local point of view. In order to do a comparison with the other countries and have a more global point of view, we thought to visualize flows between the type of meteorites and the different countries (Figure 10).

Firstly we thought to just draw the lines of direct connections between the two different nodes with a constant thickness. We then realised that a thickness depending on the mass would have been more informative. The Sankey diagram came to our mind to vizualize the flows and a nice example proposed by Bremer [8] was our inspiration. Indeed, the latter example propose a Chord diagram in the form of a bipartite Sankey diagram. We discovered a simply way to build a bipartite graph with the `viz.js` library [9]. This library takes as input parameters the primary key (meteorites' classes), the secondary key (countries) and a value that links the two keys (mass). We first considered all the countries, but we decided to filter out only the countries having a total `mass` of meteorites bigger than 500 kg and store the other countries as 'Others'.

The result of our first bipartite graph was as follow (Figure 11):

The percentages that you can see on the right of the countries correspond to the sum of all meteorites that fell on that country divided by the mass of all meteorites. If you mouse over a class of meteorite or a country, it will filter out the class selected or the country selected as you can see in figure 12. The percentages update dynamically as well.

Then we noticed that the size of the country biased our first results. In order not to give a higher weight to the biggest countries, we divided every `mass` by the area of

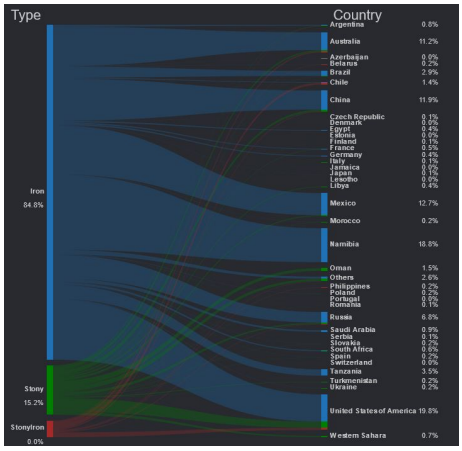


Figure 11: First bipartite graph with thickness of flow depending on the mass

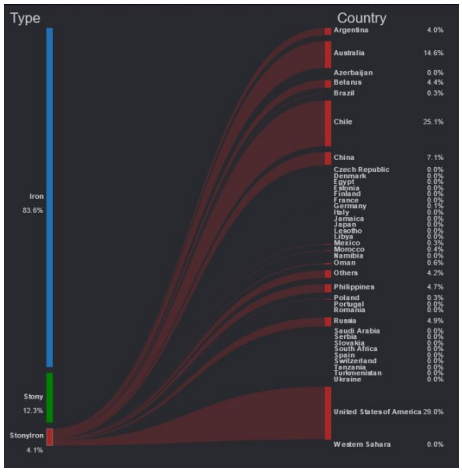


Figure 12: Bipartite graph - on mouse over *stony-iron* class

the country where it fell, obtaining then the `density` of meteorites by country in g/km^2. This time we filtered out the countries having a `density` bigger than 1 g/km^2 and store the remaining in 'Others'. The area of the countries was calculated thanks to Quantum GIS software, an open source GIS[5] software.

We ended up with the graph in figure 13. Countries as Australia, United States and China clearly diminished on importance and Russia was even incorporated to 'Others'.

G. Country statistics

The country statistics panel was from the beginning thought to appear when a country was selected in the search-bar. Hence, we tried to figure out which country's data could be interesting to visualize.

[5]Geographic Information Systems

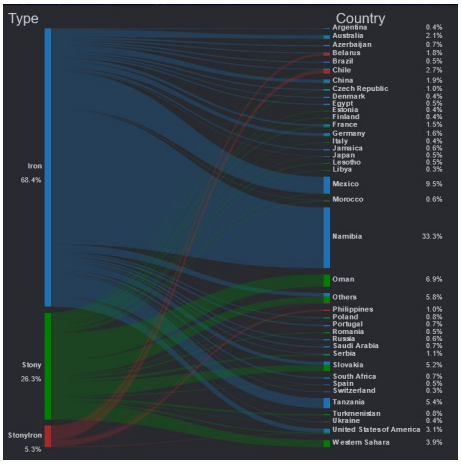


Figure 13: Final bipartite graph with thickness of flow depending on the density

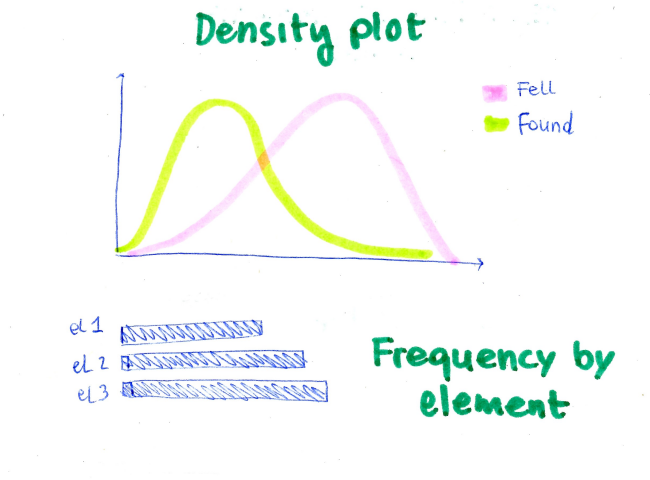


Figure 14: Sketch of density plot and frequency by element

Firstly, we decided to visualize the mass distribution of the meteorite falls and find[6] and the frequency by meteorite classes (Figure 14).

As the information of the meteorite classes was already given with the bipartite graph (Figure 13), the latter option was abandoned. Furthermore, we decided to do not take into account the differences between the $fell$ and $found$ meteorites because the number of fell meteorites was a way smaller than the found meteorites, leading the comparison between the two types difficult to interpret.

Another option that was taken into account was to visualize the three heaviest meteorite recorded that fell in

[6]Meteorites are considered falls if they can be associated with an observed fall event and finds if they cannot be connected to a recorded fall event [11].

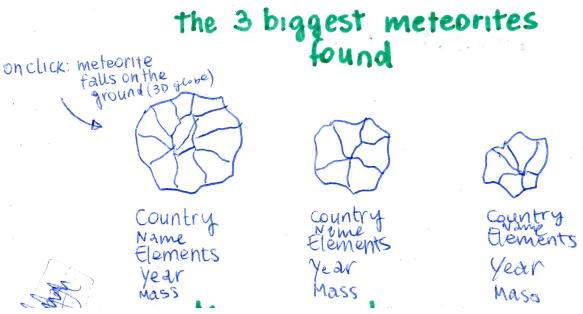


Figure 15: Sketch meteorite shape

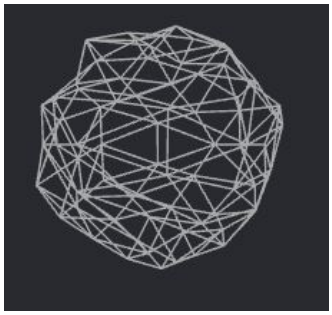


Figure 16: Heaviest meteorite with `three.js`

the selected country. A simple text saying the names and the masses of the meteorites was considered a too simple solution; hence, we decided to create a geometry that could remind a meteorite shape as illustrated in the sketch in figure 15.

We finally opted to only draw the heaviest meteorite by country giving that most of the countries had only few entries. The library `three.js` was again the best option to visualize 3D shapes. In order to draw the shape desired, we created a dodecahendrom geometry and pushed random positions to the its vertices. We obtained the shape presented in figure 16.

Our initial idea was even to click on the meteorite and see the meteorite falling, but this idea was abandoned as well for a lack of time.

Concerning the mass distribution, the first problem encountered was that the masses recorded are generally low, resulting then to very skewed distributions. In order to have a smaller range and have a more normal distribution, the mass values considered were transformed in logarithmic values.

As you can see in Figure 14, the first idea was to plot a line following the frequency of each mass. With countries having a big amount of data recorded it worked well (Figure 17), but giving that most of the countries have only few entries or even only one recorded meteorite, the visualization showed false information (Figure 18, 19).

We finally opted for a histogram of the mass values, the simplest but the clearest one. The final country statistics

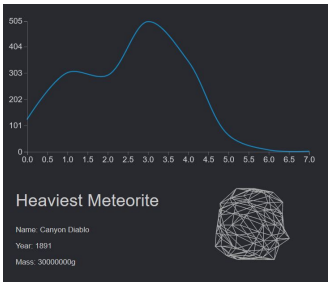


Figure 17: Density plot with many entries with `d3.js`

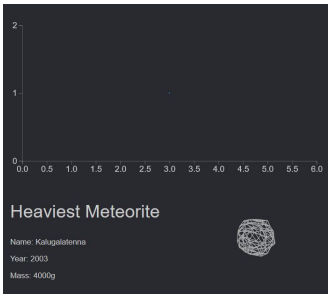


Figure 18: Density plot with one entry with `d3.js`

panel is illustrated in figure 20.

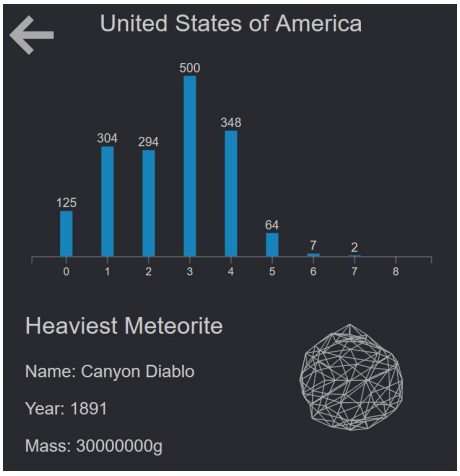


Figure 20: Final country statistics panel

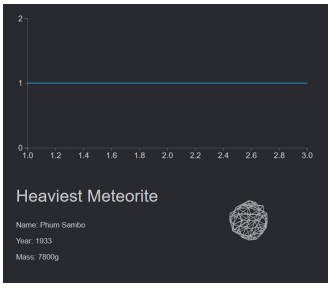


Figure 19: Density plot with two entries with `d3.js`

H. Messages

After setting up all the spatial and statistical visualizations and going deeper in the domain, we discovered many funny (and a bit sketchy) stories about meteorites in human history. This has soon gave us new ideas to represent other information and make funnier our visualization.

In fact, the International Comet Quarterly publishes a list of all incidents reported related to meteorites [10]. We took this data and adapted to make them appear next to the globe. The text does not appear all at once, but word by word. The reason behind is that it is easier to see it when a new message is displayed.

I. Final touches

Many final touches were needed in order to improve the UX.

Firstly, the texture covering the Earth depends on the user hour. From 6 p.m. until 6 a.m. the texture of the Earth globe is a night-time image (Figure 21) . The remaining hours, the texture is a day-time image with clouds (Figure 22). A black background with stars upon it has been adopted as well, leading to the illusion that our planet is being hung somewhere in the universe.

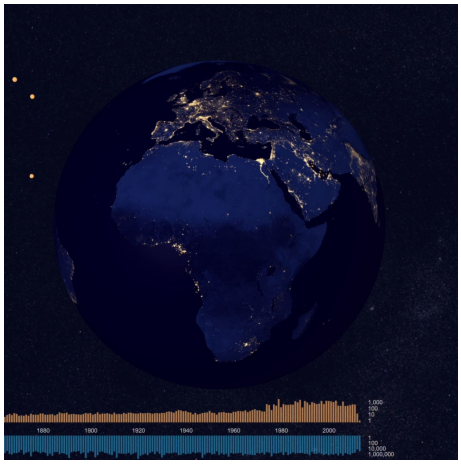


Figure 21: Night-time texture

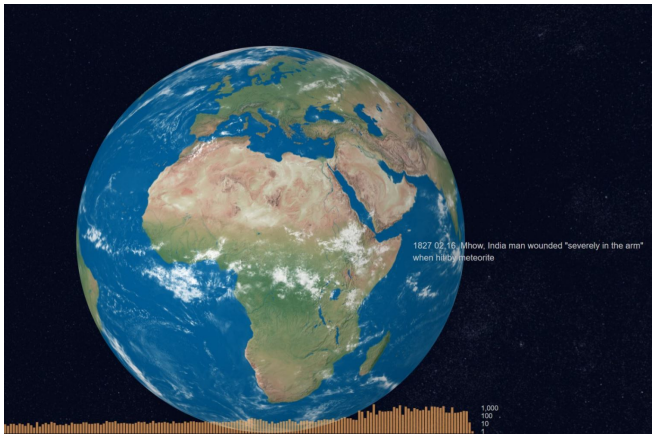


Figure 22: Day-time texture

When the user selects a country, the camera zoom-in to the country desired. This creates more interaction with the interface. The camera position is taking the extreme coordinates and the centroid of the country, that have been still calculated with Quantum GIS.

The layout of all visualization needed to be done as well. In fact, we did not think at the beginning to resize the different visualizations depending on the user screen. We then tried to resize all the elements the more dynamically as possible.

V. Improvements and limits

The time was limited, but plenty of ideas came to our mind in the process of the visualization.

The limits of our visualization are the following:

- The countries' borders are missing. It is then difficult to understand where is the country, mostly when the country have no borders touching the sea
- The meteorites falls are not filtered out by country when we zoom on a country. Hence, the meteorites falls in the neighboring countries are seen as well.
- ...

VI. Work split and peer evaluation

Rehan Mulakhel
: Web page centralizing links to the demo, code and process book. Brushing time line. Dynamic suggestion for the countries. Country to each meteorite based on coordinates.

Noemi Romano
: Data cleaning, globe visualization (partly), country statistics, meteorites classification, ISO codes for countries, getting minimum latitude and longitude for every country to facilitate the zooming on the country selected, process book.

Raja Soufi
: Globe visualization (partly) as well as falling meteorites and world camera animations/movements. Improvements to various elements such as timeline and search field, as well as linking the various parts together.

References

[1] World Atlas - geojson.

[2] GeoJSON in Three.js, Aug 2017.

[3] three.js - Javascript 3D library, Nov 2017.

[4] Mike Bostock. Brush and Zoom. https://bl.ocks.org/mbostock/34f08d5e11952a80609169b7917d4172, 2017. [Online; accessed 18-December-2017].

[5] Mike Bostock. Mike Bostocks Blocks - bl.ocks.org. https://bl.ocks.org/mbostock, 2017. [Online; accessed 17-December-2017].

[6] Mike Bostock. See-Through Globe - bl.ocks.org. https://bl.ocks.org/mbostock/6746848, 2017. [Online; accessed 17-December-2017].

[7] Mike Bostock. This Is a Globe - bl.ocks.org. https://bl.ocks.org/mbostock/ba63c55dd2dbc3ab0127, 2017. [Online; accessed 17-December-2017].

[8] Nadieh Bremer. Hacking a Chord Diagram to visualize Flows - VisualCinnamon, Dec 2017. [Online; accessed 20. Dec. 2017].

[9] Naushad Pasha Puliyambalath. NPashaP/Viz, Dec 2017. [Online; accessed 20. Dec. 2017].

[10] International Comet Quarterly. Interesting meteorite falls. http://www.icq.eps.harvard.edu/meteorites.html, 2013. [Online; accessed 18-December-2017].

[11] Michael K. Weisberg, Timothy J. McCoy, and Alexander N. Krot. Systematics and evaluation of meteorite classification. 2006.